

Trajectory Planning Using Reachable-State Density Functions

Richard Mason and Joel W. Burdick

{mason,jwb}@robotics.caltech.edu

Mechanical Engineering, California Institute of Technology
Mail Code 104-44, Pasadena, CA 91125

Abstract: *This paper presents a trajectory planning algorithm for mobile robots which may be subject to kinodynamic constraints. Using computational methods from noncommutative harmonic analysis, the algorithm efficiently constructs an approximation to the robot's reachable-state density function. Based on a multiscale approach, the density function is then used to plan a path. One variation of the algorithm exhibits time complexity that is logarithmic in the number of steps. Simulations illustrate the method.*

1 Motivation

Many mobile robots operate under non-holonomic or other kinodynamic constraints which complicate the task of planning their motions. This paper presents an algorithm for planning gross motions of many types of mobile robots in the presence of such constraints.

There is a large literature on the subject of non-holonomic and kinodynamic motion planning. Many approaches are based on a decomposition of the planning problem into two parts. In the first step, the kinematic and dynamic constraints are ignored, and the gross path of the system that avoids obstacles is determined. This "holonomic" path can be chosen manually, or by a conventional rigid body holonomic motion planner. Next, the actual control inputs that respect the constraints are determined so that the system approximately tracks the holonomic path. Based on differential geometric concepts, there are many local motion planners that compute the controls inputs which cause the vehicle to approximately track the specified trajectory (e.g. [1, 2, 3, 4]). Implicitly, this approach assumes that the vehicle is small time locally controllable, so that it is capable of locally approximating any trajectory. When this criterion is not satisfied, the success of these methods is uncertain.

The successful probabilistic roadmap motion planning paradigm [5] has also been adapted to nonholonomic motion planning (e.g. [6]). Unfortunately, a local nonholonomic motion planning problem must be

solved each time a candidate node is considered for addition to the roadmap. Lavelle and Kuffner [7] have also developed a probabilistic kinodynamic planning approach that is based on an incremental simulation of the system's dynamics. A forward simulation of the system's dynamics is used to expand a search tree. This approach circumvents the small time local controllability assumption of many other techniques, and has produced excellent simulation results. However, its probabilistic convergence may require an exponential number of computations, and other aspects of its computational complexity are still unknown.

This paper introduces a different type of algorithm for approximately solving this class of motion planning problems. Our deterministic approach is based on constructing an approximation to the vehicle's reachable set of states. We use techniques of fast Fourier transforms on Lie groups to efficiently compute these sets. This approach is motivated by the Ebert-Uphoff algorithm for solving the inverse kinematics of discretely actuated manipulators [8]. A solution path can then be constructed from knowledge of the reachable set. Like [7], this technique does not require small time local controllability, and we show that the algorithm has an attractive computational complexity. Because we use a finite set approximation to the continuous mechanics, our technique can also be used to plan the motions of "discrete" nonholonomic systems [9].

Section 2 provides an intuitive overview of the approach, focusing on the vehicle's reachable set of states. Sections 3 and 4 describe the density of reachable states concept, and algorithms for its computation. Section 5 then describes the planning method, and the algorithm is illustrated by an example. Another variation of the algorithm with tighter time and space bounds is summarized in Section 6.

2 Summary of the Approach

Most nonholonomically constrained wheeled vehicles move by the influence of periodic or quasi-periodic

controls. Hopping, walking, and swimming robots move by periodic oscillations of their driving actuators. A common theme among these systems is that their gross trajectories can be naturally broken into individual hops, steps, or undulations. The trajectories of such robots can therefore often be considered as a sequence of quasi-periodic steps, each one of which causes some change in the robot's position while returning the robot's actuators to approximately the same state at the end of the step as at the beginning. Furthermore, different combinations of the periodic inputs can be interpreted as different vehicle "gaits" that move the vehicle in different directions [10].

In the case of a wheeled robot, even though the motion appears continuous, the methods described here may still be used, by artificially discretizing the trajectory into a sequence of finite time steps with a particular choice of motion for each time step. Even if the internal configuration of the robot does not literally return to the same state at the end of each time step (for example, if wheel angle changes while rolling forward), the method is applicable as long as the robot is capable of the same set of motions in each time step.

Based on these observations, our approach discretizes the motion group $SE(D)$ ($D=2$ or 3) in which the robot operates, and plans a trajectory to any given volume element in $SE(D)$ by considering the number of reachable endpoints in that volume element and in intermediate volume elements along the trajectory. Our algorithm is largely inspired by the Ebert-Uphoff algorithm for solving the inverse kinematics of discrete multistage manipulators [8]. We adapt this algorithm to motion planning purposes, and improve upon its computational complexity. In other respects it is reminiscent of Dijkstra's algorithm as applied to path planning in a discretized configuration space by Barraquand and Latombe [11]. Like those algorithms, our method requires a memory-intensive mapping of the workspace, but generates trajectories to chosen goal points very quickly once the map is constructed.

We consider a mobile robot which moves by discrete steps, transforming its location in $SE(D)$ by any one of K motions in the finite set $A \subset SE(D)$. This may be because the robot's actuators are actually discrete, or because it is convenient to consider a restricted set of "optimal" motions at each step (e.g. a carlike robot whose optimal motions are either straight ahead, left arcs, or right arcs [12]), or solely for the sake of approximating the reachable set of states. In the case of complex kinodynamic constraints, the finite set of motions can be crafted from a forward simulation of the system for each of a finite set of inputs. This pro-

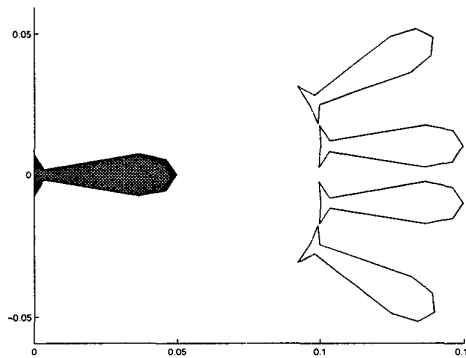


Figure 1: An example to illustrate the method. This fish robot can reach any of four positions in one tail stroke.

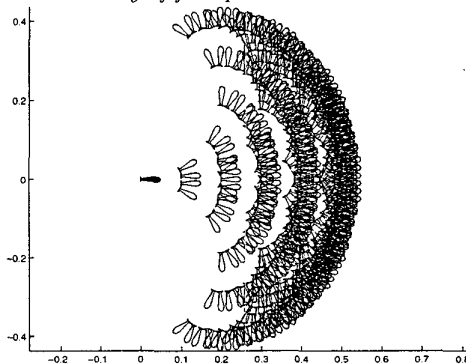


Figure 2: Positions reachable in five or fewer tail strokes.

cedure avoids the need for small time local controllability. For example, Figure 1 depicts a hypothetical robot fish which can stroke its tail to make one of four motions: turn left, move forward and sideslip left, move forward and sideslip right, or turn right. The elements of $SE(2)$ associated with these motions are $(x_1, y_1, \theta_1) = (0.0959, 0.0245, 0.5)$, $(x_2, y_2, \theta_2) = (0.1, 0.01, 0)$, $(x_3, y_3, \theta_3) = (0.1, -0.01, 0)$, and $(x_4, y_4, \theta_4) = (0.0959, -0.0245, -0.5)$. Here these values are just invented to form an example, but in practice, they would be identified by analysis, numerical simulation, or experiment with an actual system.

For a sequence of P such steps, the robot can follow K^P potential trajectories to K^P endpoints in $SE(D)$, with some of the endpoints possibly being duplicates. Figure 2 shows how the reachable configurations of the robot fish grow quickly.

3 Density Functions

We will be concerned with functions expressing the density of a cloud of points in a continuous group. Like any non-uniform density, this density function is not well-defined unless we specify the volume elements over which the density is measured. More precisely, let F be a Lie group, and $G \subseteq F$ be a relevant subset of

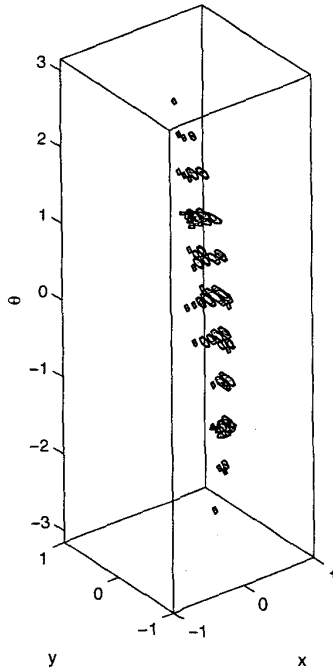


Figure 3: States in Figure 2 viewed as a density function in configuration space.

the group. We define a set-valued function V which maps every element $x \in G$ to a simply connected subset (i.e. a volume element) $V(x) \subset G$ where $x \in V(x)$. The range of V will therefore be a collection of subsets of G which cover G , but which need not be disjoint. When the covering subsets are not disjoint, $V(x)$ will indicate the generically unique volume element whose center is closest to x .

Let $n(H)$ for any subset $H \subset G$ be the number of discrete points that lie in H . And let $\|H\| \in \mathbb{R}$ be a measure of the volume of H . The density function $\rho(x) : G \rightarrow \mathbb{R}$ is defined as:

$$\rho(x) = \frac{n(V(x))}{\|V(x)\|}. \quad (1)$$

It is simplest to visualize the case where the volume elements are disjoint. In that case, the non-overlapping elements form a regular grid covering G , and for any $x \in G$, $V(x)$ is simply the element containing x .

We wish to find a reachable configuration close to a specified group element (i.e. the goal). Consequently, the volume elements should be sufficiently large so that each one contains one or more reachable points. But if the volume elements are both large and disjoint, the discretization of G is coarse, and the resulting paths will be correspondingly imprecise. We can partly avoid this problem by using non-disjoint volume elements.

To use a geographic analogy, a census taker might define the population density "at" each location, x , as the number of people within a one kilometer radius of x , and then measure and record the population density at locations spaced ten meters apart. In this case the range of V would consist of overlapping one-kilometer-radius circles with centers ten meters apart.

The choice of V determines two different kinds of spatial resolution. The first resolution involves the number of volume elements in a particular volume of G . The second resolution is the size of the volume elements. If the volume elements are disjoint, these two resolutions are essentially the same. The point of overlapping volume elements is that the first resolution can remain fine while the second one is made coarse.

We start with a discretization V_0 where the volume elements are disjoint, and then construct coarser discretizations as necessary. For example a coarser discretization V_1 on G can be defined as follows. Let $V_0(G)$ be the range of V . Then for any $x \in G$,

$$V_1(x) = \bigcup \{H \in V_0(G) | H \cap V(x) \neq \emptyset\}.$$

If H borders $V(x)$, then we say that its intersection with $V(x)$ is nonempty. We can iterate this coarsening procedure. For $k > 1$,

$$V_k(x) = \bigcup \{H \in V_{k-1}(G) | H \cap V_{k-1}(x) \neq \emptyset\}. \quad (2)$$

If the volume elements in the range of V_0 have uniform volume, then the volume of the elements in the range of V_k will increase exponentially with k .

4 Computing Density Functions

Recall our premise that the robot transforms its location in $SE(D)$ by one of K possible motions with each step. There are K^P possible sequences of P steps, taking the robot to K^P endpoints in $SE(D)$. We define ρ^P such that $\rho^P(g)$ for a configuration $g \in SE(D)$ is the number of endpoints in the volume element $V(g)$ centered on g divided by the volume of $V(g)$. (See Fig. 3.)

For small P , we can compute ρ^P by simply enumerating the K^P points that can be reached. Obviously this method becomes unfeasible as P grows large. For larger P , the density function ρ^P can be found by taking the convolution of two known density functions representing smaller numbers of steps.

$$\rho^P(H) = (\rho^Q * \rho^R)(H) \quad (3)$$

$$= \int_{SE(D)} \rho^Q(\mathcal{H}) \rho^R(\mathcal{H}^{-1}H) d(\mathcal{H}) \quad (4)$$

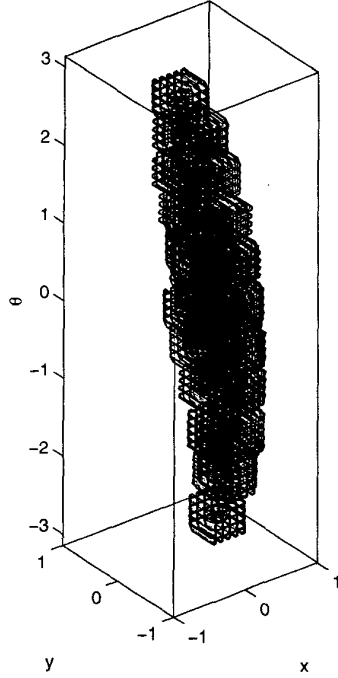


Figure 4: Density from Figure 3 on a much coarser scale.

where $H, \mathcal{H} \in SE(D)$ and $Q + R = P$.

Since $\rho^k * \rho^k = \rho^{2k}$, we can produce ρ^P with a number of convolutions in $\mathcal{O}(\log P)$ operations.

Since ρ^1 has at most K non-zero elements, straightforward convolution of ρ^1 with any other function can be performed in $\mathcal{O}(NK)$ time. If K is not large¹, this may actually be the most sensible way in which to proceed. However even if K is not large, this method has the drawback that ρ^P can only be computed by serially convolving ρ^1 with $\rho^1, \rho^2, \dots, \rho^{(P-1)}$, requiring $\mathcal{O}(P)$ convolutions.

Performing a convolution by numerical integration of Eq. (4) should require $\mathcal{O}(N^2)$ operations, where N is the total number of volume elements in the discretization of $SE(D)$. For approximation error to be kept low, N must be quite large, even for $D = 2$ and especially for $D = 3$, so naive convolution becomes prohibitively expensive. Fortunately, a faster method of convolution is available.

4.1 The Fourier Transform

It is common knowledge that in \mathbb{R}^n , the Fourier transform of two convolved functions is simply the

¹If K is large, in particular if K is an appreciable fraction of N , then this method may be computationally prohibitive. The more sophisticated methods of convolution described later will definitely be superior if K is proportional to N^γ where $\gamma > \frac{1}{3}$.

N	Total number of samples on $SE(2)$	$\mathcal{O}(S^3)$
N_r	Number of samples on $SO(2)$	$\mathcal{O}(S)$
N_R	Number of samples on \mathbb{R}^2	$\mathcal{O}(S^2)$
N_p	Number of samples on p interval	$\mathcal{O}(S)$
N_u	Number of samples on $[0, 2\pi)$	$\mathcal{O}(S)$
N_F	Total number of harmonics	$\mathcal{O}(S^2)$

Table 1: Resolution measurements in $SE(2)$.

product of their individual Fourier transforms.

$$\mathcal{F}(f_1 * f_2) = \mathcal{F}(f_1)\mathcal{F}(f_2) \quad (5)$$

The fast Fourier transform (FFT) algorithm provides an efficient way to numerically approximate a function's Fourier transform. Since the FFT, the multiplication of the resulting Fourier transforms, and the inverse FFT can each be performed in subquadratic time, this is fast way to perform convolution in \mathbb{R}^n . To generalize these concepts to $SE(D)$, we apply concepts from the field of noncommutative harmonic analysis. The remainder of this section summarizes material from [8].

We can express the matrix elements of the Fourier transform of a function $f(\mathbf{r}, \theta)$ on $SE(2)$ by [8]

$$\hat{f}_{mn}(p) = \int_{\mathbf{r} \in \mathbb{R}^2} \int_{\theta=0}^{2\pi} \int_{\psi=0}^{2\pi} f(\mathbf{r}, \theta) \times e^{in\psi} e^{i(\mathbf{p} \cdot \mathbf{r})} e^{-im(\psi-\theta)} d^2r d\theta d\psi. \quad (6)$$

In practice we use a band-limited approximation of the Fourier transform and only compute elements with $|m|, |n| < S$ for some $S \in \mathbb{Z}^+$. The group $SE(2)$ is discretized with the number of samples described in Table 1. The Fourier transform can be carried out numerically in the following way. First, assuming that $f(\mathbf{r}, \theta)$ is sampled on a Cartesian grid of $\mathbf{r} \in \mathbb{R}^2$ values, the integration over \mathbb{R}^2 is:

$$f_1(\mathbf{p}, \theta) = \int_{\mathbb{R}^2} f(\mathbf{r}, \theta) e^{i(\mathbf{p} \cdot \mathbf{r})} d^2r \quad (7)$$

using the usual FFT in $\mathcal{O}(N_R N_r \log(N_r))$ time. Then we interpolate from the Cartesian grid to sample $f_1(\mathbf{p}, \theta)$ on a polar-coordinate grid of \mathbf{p} values. This set of $f_1(p, \psi, \theta)$ values can be found in $\mathcal{O}(N_R N_r) = \mathcal{O}(N)$ time assuming that the interpolation of each individual grid point can be done in constant time. (See Table (1) for definitions of N, N_R , et cetera.) Then the usual one-dimensional FFT can be used to integrate, first along the θ dimension in $\mathcal{O}(N_r N_R \log(N_R))$ time

$$f_2^{(m)}(p, \psi) = \int_{SO(2)} f_1(p, \psi, \theta) e^{im\theta} d\theta \quad (8)$$

and along the ψ dimension in $\mathcal{O}(N_R N_P N_u \log(N_u))$ time

$$\hat{f}_{mn}(p) = \int_0^{2\pi} \left[f_2^{(m)}(p, \psi) e^{-im\psi} \right] e^{in\psi} d\psi. \quad (9)$$

The entire process takes $\mathcal{O}(N \log(N))$ time. The recovery of a function from its band-limited Fourier transform also takes $\mathcal{O}(N \log(N))$ time.

$$f(\mathbf{r}, \theta) = \frac{1}{2\pi} \int_0^\infty p dp \sum_{m,n=-S}^S \left[\hat{f}_{mn}(p) \times \int_0^{2\pi} e^{i(m-n)\psi} e^{-i\mathbf{r} \cdot \mathbf{p}} e^{-im\theta} d\psi \right] \quad (10)$$

The Fourier transform on $SE(2)$ has the crucial convolution property [13]

$$\mathcal{F}(f * g) = \mathcal{F}(g) \mathcal{F}(f), \quad (11)$$

$$\mathcal{F}(f * g)_{mn}(p) = \sum_{k=-S}^S \hat{g}_{mk}(p) \hat{f}_{kn}(p). \quad (12)$$

Thus, the convolution of functions f and g on $SE(2)$ can be performed by finding the Fourier transforms in $\mathcal{O}(N \log(N))$ time, multiplying the Fourier matrices, and performing the inverse Fourier transform in $\mathcal{O}(N \log(N))$ time. The most computationally expensive step is matrix multiplication, which takes $\mathcal{O}(NN_R) = \mathcal{O}(N^{4/3})$ time if the “obvious” method of matrix multiplication is used. This bound can be slightly improved by using more advanced matrix multiplication algorithms.² In any case, for large N this method is clearly a drastic improvement over $\mathcal{O}(N^2)$ straight-forward numerical convolution.

5 Trajectory Planning

This section reviews the Ebert-Uphoff Algorithm that we have adapted to motion planning. Our enhancements to the algorithm are then summarized.

5.1 The Ebert-Uphoff Algorithm

Let V_0^j denote the discretization of $SE(D)$ chosen for a j step density function³. Suppose that the density functions ρ^j and their associated discretizations V_0^j for $1 \leq j \leq P$ are known. To find a trajectory which ends as closely as possible to the goal

²The matrix multiplication step will require $\mathcal{O}(N^{(\gamma+1)/3})$ computations, where $\gamma = 3$ for “standard” matrix multiplication, but $\gamma = \log_2 7 \approx 2.81$ using Strassen’s algorithm [8].

³Generally, the discretizations V_0 and V_0^j are the same. However, we allow for the option that one may choose to resample $SE(D)$ for purposes of refining the density function ρ^j .

$g_{\text{des}} \in SE(D)$ in P steps, one sequentially chooses the j^{th} motion step to maximize the reachable state density function of the remaining $P - j$ steps around the goal. That is, the first step g_1 is chosen from the set of K allowable motions in $SE(D)$ to maximize $\rho^{(P-1)}(g_1^{-1} g_{\text{des}})$. With g_1 fixed, g_2 is chosen to maximize $\rho^{(P-2)}(g_2^{-1} g_1^{-1} g_{\text{des}})$. We go on to choose the j^{th} step g_j to maximize $\rho^{(P-k)}(g_j^{-1} \dots g_2^{-1} g_1^{-1} g_{\text{des}})$. After $P - 1$ steps, the final step g_P is chosen in order to minimize a distance measure

$$e = d(g_{\text{des}}, g_1 g_2 \dots g_P) \quad (13)$$

This scheme for finding a trajectory which approximately reaches the goal in P steps is identical to the Ebert-Uphoff algorithm for solving the inverse kinematics of a discretely actuated P -link manipulator [8]. It returns an approximate solution in $\mathcal{O}(KP)$ time.

5.2 Implicit Storage of Paths in Graph

Because this algorithm starts with a goal reachable in P steps, makes a transformation to a point reachable in $P - 1$ steps, then $P - 2$ steps and so on, it is vaguely reminiscent of Dijkstra’s algorithm as applied to a graph whose nodes are reachable volume elements of a configuration space [11]. We can increase the resemblance by explicitly constructing such a graph.

For every configuration g_0 where $\rho^j(g_0)$ is non-zero, let $v = \{g \mid V^j(g) = V^j(g_0)\}$. To construct the graph, search for the $g \in v$ and $h \in A$ which maximize $\rho^{j-1}(h^{-1}g)$, then store a pointer between $V(g)$ and $V(h^{-1}g)$. This step requires $\mathcal{O}(2^D KNP)$ time for P density functions. Alternatively, instead of searching over both g and h , one can specify a “representative” value of $g \in v$; in this case the graph construction takes $\mathcal{O}(KNP)$ time. In either case, by following edges of the constructed graph instead of explicitly checking K density values at each step of the trajectory, paths can be found in $\mathcal{O}(P)$ rather than $\mathcal{O}(KP)$. However, a degree of inaccuracy is introduced into the trajectory because of round-off error.

When such an explicit graph is used, the method is still distinct from the traditional application of Dijkstra’s algorithm in at least two ways. Dijkstra’s algorithm chooses steps backwards from the goal to the trajectory, whereas this method chooses steps in a forward fashion. This means *inter alia* that while both methods require $\mathcal{O}(P)$ time to construct the whole path, if only the next step in the path is required then this method can find it in $\mathcal{O}(1)$ time. Also, instead of associating a single pointer with each volume element and implicitly storing only the shortest path

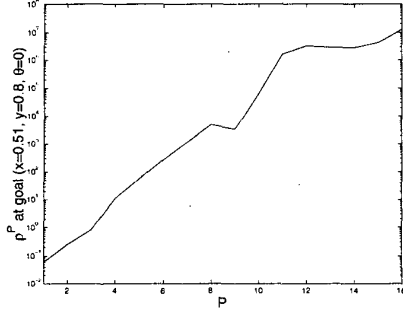


Figure 5: The density ρ^P in the vicinity of a particular goal, as a function of P . The path length P can be chosen so that ρ^P is sufficiently high. Often, ρ^P increases with increasing P , so a longer path may yield a better approximation to the goal.

to each volume element, up to P pointers and implicitly up to P successful trajectories are stored for every volume element. With this construction, one can either choose the shortest path which approximately reaches the goal, or possibly choose a longer path which reaches the goal more exactly.

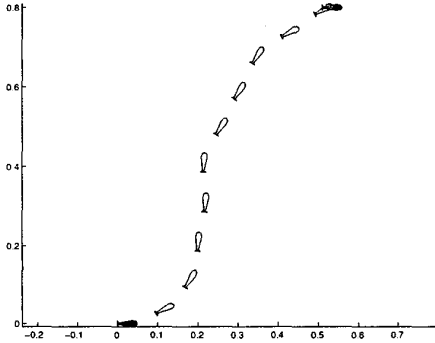


Figure 6: The algorithm finds a 10-step path to the goal $(x, y, \theta) = (0.51, 0.8, 0)$. While the final position $(x, y, \theta) = (0.492, 0.783, 0.5)$ is not quite right, this is roughly the shortest path with possibly acceptable error.

Example. Figs. 5-8 show the output from an implementation of this algorithm as applied to the fish robot of Fig. 1 trying to reach the goal $(x, y, \theta) = (0.51, 0.8, 0)$. Figs. 6-8 illustrate the trade-off between path length and final goal accuracy.

5.3 Multiscale Densities

This section introduces a multiscale extension of the basic algorithm. Recall that ρ_k^j denotes the density function ρ^j defined at the coarser scale V_k^j . To motivate the multiscale approach, note that each density function ρ^j is associated to a discretization V^j . The different discretizations V^1, \dots, V^P need not be identical. The algorithm, as described so far, neglects the

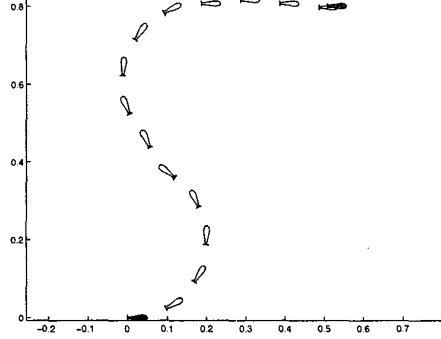


Figure 7: Higher accuracy can be obtained by varying P to obtain a more dense ρ^P . This 14-step path ends closer to the goal, at $(x, y, \theta) = (0.489, 0.797, 0)$.

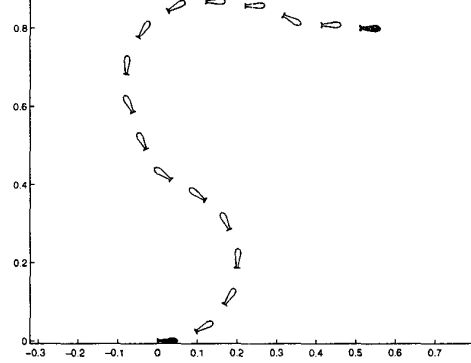


Figure 8: A 16-step path reaches the goal configuration almost perfectly: $(x, y, \theta) = (0.513, 0.798, 0)$.

possibility that at the j^{th} step g_j , the density function $\rho^{(P-j)}(g_j^{-1} \dots g_2^{-1} g_1^{-1} g_{\text{des}})$ is zero for all K possible choices of g_j . In this case the goal g_{des} lies in a volume element which is not considered reachable.

It might then be said that the resolution of V^j was chosen too fine. Since we wish to reach the closest configuration to g_{des} , we should have chosen the volume elements big enough so that the one which contains g_{des} also contains some reachable point. Unfortunately, in advance of knowing g_{des} , it is not necessarily possible to know how fine a resolution is too fine.

We now adapt the algorithm to use density functions at a coarser resolution when necessary. Recall that according to the coarsening scheme outlined earlier, for $k \geq 1$,

$$V_k^j(x) = \bigcup \{H \in V_{k-1}^j(G) \mid H \cap V_{k-1}^j(x) \neq \emptyset\}. \quad (14)$$

We want to find the smallest k such that $\rho_k^{(P-j)}(g_j^{-1} \dots g_2^{-1} g_1^{-1} g_{\text{des}})$ is non-zero for some valid choice of g_j . Then we choose g_j to maximize this value.

In the worst case, it might be necessary to consider

$\mathcal{O}(\log(N))$ scales.⁴ All $\mathcal{O}(\log(N))$ scales can be constructed from the finest scale in $\mathcal{O}(N \log(N))$ time, which is dominated by the time required to construct the density function at the finest scale in the first place. To store the density function at all $\mathcal{O}(\log(N))$ scales would require $\mathcal{O}(N \log(N))$ space. But not all of this information need be retained. The construction of the density functions at coarser scales only requires knowledge of the next finer scale. We only need to record for each volume element the finest scale at which the density function assumes non-zero value there, and that non-zero value. Consequently, the multiscale algorithm requires $\mathcal{O}(N)$ space.

Assume that the multiscale density functions are precomputed, and that the finest scale at which each volume element has a non-zero density is known. Then the algorithm can be employed as before, with the proviso that arguments are evaluated at the finest scale which displays non-zero density. Since this can be done in constant time for each argument, the worst-case performance of the modified algorithm is still $\mathcal{O}(KP)$, or $\mathcal{O}(P)$ if a graph is used to store trajectories.

5.3.1 Error Bounds

The worst-case error can be bounded at the outset by the size of the volume element $V_k^P(g_{\text{des}})$ for the smallest value of k such that $\rho_k^P(g_{\text{des}})$ is non-zero. After j steps have already been taken, the remaining expected error is bounded by the size of $V_k^{(P-j)}(g_j^{-1} \dots g_2^{-1} g_1^{-1} g_{\text{des}})$ for the smallest k such that $\rho_k^{(P-j)}(g_j^{-1} \dots g_2^{-1} g_1^{-1} g_{\text{des}})$ is non-zero. And of course, when the algorithm is complete the actual error is $e = d(g_{\text{des}}, g_1 g_2 \dots g_P)$.

6 A Logarithmic Algorithm

We now present a variation of the algorithm with logarithmic time complexity. This version is based on the observation that if the densities $\rho^j, \rho^{j+1}, \dots, \rho^{2j}$ are known, then simple addition yields the function $\rho^{[j, 2j-1]}$, which is the density of endpoints reachable by trajectories of at least j but no more than $2j - 1$ steps. Convolution of $\rho^{[j, 2j-1]}$ with ρ^j yields

⁴We have defined a series of scales whose individual volume elements increase geometrically in size. Only $\mathcal{O}(\log(N))$ scales can be defined before reaching the coarsest scale, where the density function is uniform across the whole environment. We could instead have coarsened the scale more gradually, say linearly, and defined the density function at $\mathcal{O}(N)$ different scales. But the complexity of this multiscale algorithm would rise to $\mathcal{O}(KNP)$ in time, and the time required to construct the different density functions in the first place would be $\Omega(N^2)$.

the function $\rho^{[2j, 3j-1]}$, and convolution of $\rho^{[j, 2j-1]}$ with ρ^{2j} results in $\rho^{[3j, 4j-1]}$. The functions $\rho^{[2j, 3j-1]}$ and $\rho^{[3j, 4j-1]}$ can be added to find $\rho^{[2j, 4j-1]}$. Finally convolution of ρ^{2j} with itself produces ρ^{4j} . Thus, three convolutions and one addition extends our knowledge of ρ^j, ρ^{2j} , and $\rho^{[j, 2j-1]}$, to ρ^{4j} and $\rho^{[2j, 4j-1]}$. A further addition obtains $\rho^{[2j, 4j]}$. Iteration of this procedure produces the sequence of functions $\rho^{[1, 2]}, \rho^{[2, 4]}, \rho^{[4, 8]}, \rho^{[8, 16]}, \dots, \rho^{[P/2, P]}$ in time and space logarithmic in P .

Armed with these functions, a trajectory to $g_{\text{des}} \in SE(D)$ is planned as follows. Choose a trajectory length P large enough so that the density $\rho^{[P/2, P]}(g_{\text{des}})$ is sufficiently high; the higher it is, the more closely the goal is likely to be reached. Using a function maximization algorithm, search for an element $h \in SE(D)$ which maximizes the product $\rho^{[P/4, P/2]}(h) \rho^{[P/4, P/2]}(h^{-1} g_{\text{des}})$. Having found such an element h , search for a path of no more than $P/2$ steps which approximates h , and another path of no more than $P/2$ steps which approximates $h^{-1} g_{\text{des}}$. In this manner, by recursively finding the midpoint of each unknown path segment, we eventually find P points along the path, which describe an entire trajectory which approximately reaches g_{des} .

This approach exploits the exponential properties of convolution better than the “linear” Ebert-Uphoff algorithm, and therefore only requires the computation and storage of $\mathcal{O}(\log P)$ density functions. The main drawback is the use of the function-maximization search. Depending on the details of the system, it may be difficult to predict the time τ required by this step. In the theoretical worst case, if it is necessary to check every possible volume element to find the maximum product of densities, then τ could be $\mathcal{O}(N)$. But if the density functions of the system are fairly well-behaved then in practice τ will be manageably small. For example if direction set maximization methods are usable then τ should only be quadratic in the dimension of the configuration space.

Using a single processor, the path-finding procedure takes $\mathcal{O}(\tau P)$ time to find a trajectory. However, the algorithm lends itself to parallelization. If $\mathcal{O}(P)$ processors are available, then the trajectory can be found in $\mathcal{O}(\tau \log(P))$ time.

7 Discussion

Table 2 summarizes the time and storage complexities of the variations of the algorithm, and a comparison to [11]. Note however that while our algorithm is roughly comparable to [11] for single processor im-

Method	Time to Construct Map	Space	Time to Plan Path
Barraquand and Latombe	$\mathcal{O}(NK)$	$\mathcal{O}(N)$	$\mathcal{O}(P)$
Ebert-Uphoff	$\mathcal{O}(N^{(\gamma+1)/3}P)$ or $\mathcal{O}(NKP)$	$\mathcal{O}(NP)$	$\mathcal{O}(KP)$
Multiscale Ebert-Uphoff	$\mathcal{O}(N^{(\gamma+1)/3}P)$ or $\mathcal{O}(NKP + NP \log N)$	$\mathcal{O}(NP)$	$\mathcal{O}(KP)$
Multiscale E-U plus graph	$\mathcal{O}(N^{(\gamma+1)/3}P + NKP)$	$\mathcal{O}(NP)$	$\mathcal{O}(P)$
Log method	$\mathcal{O}(N^{(\gamma+1)/3} \log P)$	$\mathcal{O}(N \log P)$	$\mathcal{O}(\tau P)$
Log method plus graph	$\mathcal{O}(N^{(\gamma+1)/3} \log P + N\tau \log P)$	$\mathcal{O}(N \log P)$	$\mathcal{O}(P)$
Log method with $\mathcal{O}(P)$ processors	$\mathcal{O}(N^{(\gamma+1)/3}(\log P)/P)$	$\mathcal{O}(N \log P)$	$\mathcal{O}(\tau \log P)$
Log method, graph, $\mathcal{O}(P)$ processors	$\mathcal{O}(N^{(\gamma+1)/3}(\log P)/P + N\tau(\log P)/P)$	$\mathcal{O}(N \log P)$	$\mathcal{O}(\log P)$

Table 2: Time and space bounds, where N is the number of volume elements, P is the number of trajectory steps, K is the number of possible motions at each step, and the constants τ and γ depend upon the chosen methods of function maximization and matrix multiplication. Although the Barraquand/Latombe algorithm has the lowest time and space requirements, it does not have all the same utility as the algorithms presented here; notably it does not allow task-by-task trade-offs between path length and accuracy.

plementation, it does not require the small time local controllability constraint, and has better control over the terminal error bounds.

8 Conclusion

This paper introduced a gross motion planning algorithm applicable to many mobile robots which are subject to kinodynamic constraints, or whose motions are restricted to a finite set, and which may not be small time locally controllable. Variants of the algorithm described here require intensive memory and considerable precomputation, but once the necessary functions have been computed, particular paths can be planned very quickly. By using fast Fourier transform techniques, we can perform the necessary precomputation in a reasonable time. Another advantage of the algorithm is that the robot can plan either a short path that approximately reaches the goal, or a more convoluted path that hits the goal configuration more exactly. The trade-off between path length and goal accuracy can be made on a task-by-task basis and does not need to be determined at the precomputation stage.

References

- [1] G. Lafferriere and Hector J. Sussmann. A differential geometric approach to motion planning. In Z. Li and J. F. Canny, editors, *Nonholonomic Motion Planning*, pages 235–270. Kluwer, 1993.
- [2] R.M. Murray and S.S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38:700–716, 1993.
- [3] Leonid Gurvits. Averaging approach to nonholonomic motion planning. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2541–2546, 1992.
- [4] R.M. Murray and S.S. Sastry. Steering nonholonomic systems in chained form. *IEEE Transactions on Control and Decision Conference*, pages 1121–1126, 1991.
- [5] L. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.
- [6] P. Svestka and M.H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Proc. IEEE Int. Conf. on Robotics and Automation*, Nagoya, Japan, Apr. 1995.
- [7] S.M. Lavalle and J.J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proc. Workshop on Algorithmic Foundations of Robotics*, Hanover, NH, Mar. 2000.
- [8] Gregory S. Chirikjian and Alexander B. Kyatkin. *Engineering Applications of Noncommutative Harmonic Analysis*. CRC Press, Boca Raton, 2001.
- [9] M. Ceccarelli, A. Marigo, S. Piccinocchi, and A. Bicchi. Planning motions of polyhedral parts by rolling. *Algorithmica*, 26:560–576, 2000.
- [10] J.P. Ostrowski and J.W. Burdick. The mechanics and control of undulatory locomotion. *Int. J. Robotics Research*, 17(7):683–701, July 1998.
- [11] J. Barraquand and J.C. Latombe. On non-holonomic mobile robots and optimal maneuvering. *Revue d'Intelligence Artificielle*, 3(2), 1989.
- [12] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2), 1990.
- [13] A.B. Kyatkin and G.S. Chirikjian. Synthesis of binary manipulators using the fourier transform on the euclidean group. *Journal of Mechanical Design*, 121:9–14, March 1999.